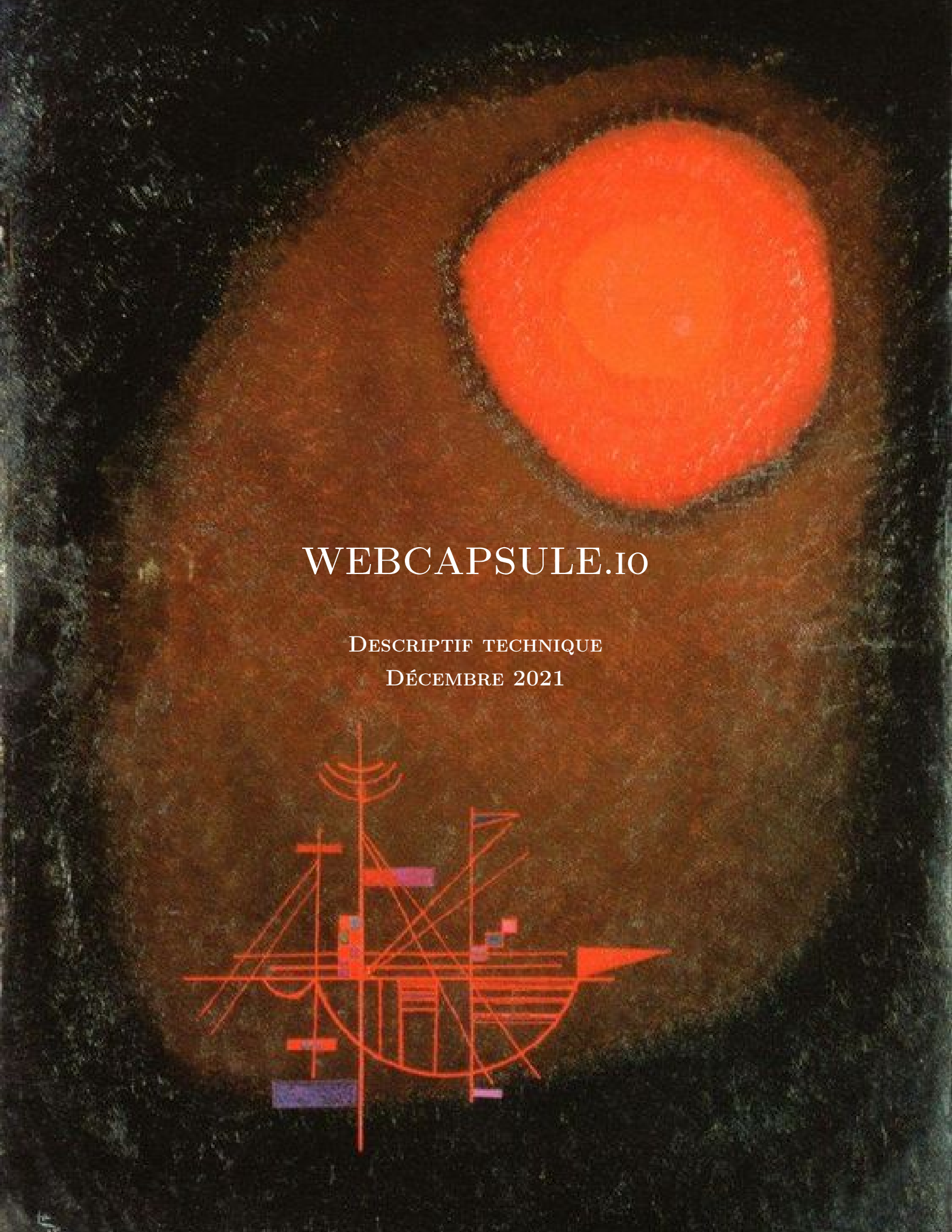
WEBCAPSULE.io

Descriptif technique

Décembre 2021



Sommaire

Introduction

Ce document est une spécification technique provisoire du service webcapsule. Il explicite les problèmes auxquels le service souhaite répondre, les directions techniques empruntées et les utilisations préconisées.

Le problème général

Le développement de la *WebCapsule* part d'un constat : la mise en production d'une application web est un travail souvent fastidieux, long et difficile à réaliser sur la durée. Cela provient de plusieurs facteurs :

- la multiplicité des offres des fournisseurs de cloud
- la multiplicité des configurations logiciels
- les exigences croissantes sur les performance d'une application web
- les exigences sur la sécurité et le suivi qu'elles impliquent
- l'accumulation régulière de dettes techniques sur les serveurs de production

Ainsi, le déploiement d'une application web demande un savoir-faire technique élevé dans un domaine en mouvement rapide. Les compétences sont coûteuses ; les entorses aux bonnes pratiques conduisent, tôt ou tard, à des pannes ou à des vulnérabilités qui ne le sont pas moins.
Nous pensons qu'une plateforme centralisée de déploiement des applications web permettrait de résoudre, de façon efficace, ces difficultés, d'améliorer la robustesse et la sécurité des applications, de permettre un gain de temps dans la gestion du déploiement.

Les avantages

Les avantages suivantes peuvent être attendues par la construction d'une plateforme centralisée :

— Un gain de temps, lors du déploiement et du développement
— Un gain de robustesse, sur la pérennité de l'infrastructure
— Un gain de sécurité, en cas d'apparition d'une vulnébarité (données pérennes, serveurs surveillés,...)
— Un gain de confort, sur les outils utilisés (espace centralisé, outils de monitoring à la dernière mode, ...)

Cheminement

Pour mieux décrire la webcapsule, nous nous proposons de découper la vie d'une application en trois parties. Nous traitons d'abord l'étape de son déploiement, ensuite la façon dont les serveurs sont gérés, enfin les outils mis en oeuvre pour observer l'application.

Étape 1

Le déploiement continu

1.1 Le premier déploiement

1.1.1 Le problème

Lors d'un premier déploiement, l'équipe de développement doit louer un serveur, installer les dépendances correctes, s'engager sur l'architecture des services. Cette étape est coûteuse et souvent réalisée tardivement. Cela conduit à omettre certaines bonnes pratiques dont la mise en place est reléguée à plus tard.

1.1.2 Proposition webcapsule

Nous mettons en place par défaut une *stack* spécifique à la technologie choisie. Il suffit de se brancher à la *webcapsule* et l'ensemble des dépendances nécessaires seront préconfigurées au plus proche du code. Ce déploiement flexible et standardisé permet aussi une sortie facile du service, en cas d'insatisfaction ; il n'implique pas d'engagement technologique.

Point d'entrée

Deux solutions sont possibles pour brancher un projet à la webcapsule.

- Si le projet est déjà en cours de développement, le point d'entrée est un dépôt git. L'utilisateur se connecte à son compte (gitlab, github, autre...) et indique ensuite le dépôt que la webcapsule doit prendre en charge.
- Pour les technos wordpress et drupal, il est aussi possible de commencer un projet de zéro, sans dépôt git.

Exemples

PHP - laravel : Soit un projet git gérant une application Laravel. L'utilisateur connecte son dépôt git à la webcapsule. Celle-ci déploie ensuite la pile suivante (cf 2.1.2) : apache, php-fpm, memcache sur les serveurs, se connectant à une base de données distantes.

Drupal : Soit un projet drupal non encore lancé. L'utilisateur se connecte à la *webcapsule* et choisit la technologie Drupal. Un projet vide est créé. Celui-ci tourne dans un environnement incluant apache, php-fpm, memcache et une base de données distantes. (cf 2.1.2)

1.1.3 Les avantages

- Rapidité du premier déploiement
- Infrastructure flexible dès le premier déploiement
- Bonnes pratiques mises en place automatiquement

1.2 Le déploiement continu

1.2.1 Le problème

Le déploiement continu vise à passer de façon fluide d'un code développé en local à un code déployé en production. Cela inclut, en général, les trois points suivants :

— disposer d'un environnement iso-prod pour anticiper les bugs sur l'environnement de production
— disposer d'un environnement de production à plusieurs instance pour déployer sans coupure
— disposer d'un outil pour déployer à distance, sans se connecter directement au serveur

1.2.2 Proposition webcapsule

Trois environnements différents

La *webcapsule* maintient trois environnements distincts. Un environnement de *staging* (aussi nommé bac-à-sable) permet de tester manuellement des modifications, de réaliser des tests d'intégrations et de fonctionnalités. Il a sa propre base de données.
Avant la mise en production, un environnement *iso-prod*, qui reprend au plus près l'environnement de production, permet de vérifier la compatibilité des modifications avec l'environnement de production.
L'environnement de *production* fonctionne lui de façon isolée, sans interférence avec les autres environnements.

En pratique

L'environnement de staging et l'environnement de production correspondent à une branche d'un dépôt git. Il revient à l'utilisateur de désigner quelle branche correspond à quel environnement.
L'environnement iso-prod est actualisé à chaque ajout d'un commit sur la branche de production. Si l'utilisateur est satisfait de ce qu'il observe sur l'environnement iso-prod, il n'a, pour déployer son code, qu'à se connecter à la *webcapsule* et à cliquer sur le bouton de déploiement. La non-interruption de l'application est garantie par l'infrastructure en cluster (cf 2.1.2).

1.2.3 Avantages

— Standardisation du déploiement
— Erreurs identifiées plus en amont
— Rollback possibles en cas d'erreurs

Le déploiement continu conduit à un gain de temps, non seulement au moment du déploiement, mais aussi lors du développement, plus serein et plus fluide, de l'application.

1.3 Déploiement continu et base de données

1.3.1 Le problème

Lors du déploiement continu, la majorité des difficultés est liée aux migrations de la base de données. Les modifications de structure, les changements dans le modèle de la dB, l'introduction de champs et de documents à conserver entre staging et production causent souvent, lors de leur mise en production, des incompatibilités sur le format des données, des erreurs, des bugs. Ce problème recouvre plus précisément deux difficultés :

— La première est de capter les modifications de structure de la base de données pertinentes à déporter en production
— La seconde est d'effectuer ces modifications sans coupure du service lors de la mise en production.

1.3.2 Proposition webcapsule

Il n'existe pas aujourd'hui d'outil sur le marché abordant ce problème. La webcapsule souhaite construire un outil capable de :

— capter les modifications pertinentes (dans la structure des données ou dans les données même) et proposer à l'utilisateur celles à exclure ou à conserver
— orchestrer une migration de la base de données sans coupure, grâce au concept de vues et de hooks

Une partie des problèmes peut aussi être repérée en amont par l'utilisation de l'environnement iso-prod.

1.3.3 Avantages

— diminuer le nombre d'erreurs sur la base de données
— augmenter la fiabilité et la sécurité du service

Étape 2

La gestion de l'infrastructure

2.1 La vie courante de l'infrastructure

2.1.1 Le problème

Du serveur isolé au réseau kubernetes, les possibilités d'hébergement d'une application sont multiples. Le choix détermine les performances de l'application, sa disponibilité et le coût de sa gestion. En effet, la gestion de l'infrastructure suppose d'être capable de maintenir celle-ci à jour et en ordre avec les dernières bonnes pratiques, notamment en terme de sécurité. L'enjeu est alors de proposer une infrastructure adaptée à la taille de l'application, pour maximiser les performances tout en simplifiant sa maintenance.

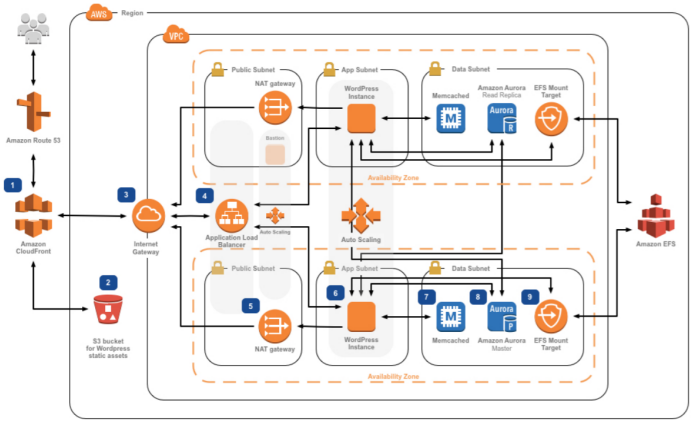
2.1.2 Proposition webcapsule

Un cluster de serveur

Chaque environnement de production est déployé, pour la partie serveur, sur AWS Fargate avec au moins deux instances. La base de données est distante et consiste, à son tour, en un cluster de serveur. Les services se situent derrière un loadBalancer et un CDN est disponible pour accélérer les requêtes en cache. Enfin, un système de mise à l'échelle automatique permet à l'infrastructure de s'adapter à la charge.
Par ailleurs, la webcapsule souhaite, à terme, s'interfacer avec différents cloud-providers.

Exemple - wordpress

L'image suivante décrit l'architecture qui permet de faire tourner une application wordpress sur aws.



Un application loadbalancer redirige le trafic vers différents conteneurs docker, tournant sur différentes zones géographiques. Chaque conteneurs partage un système de fichier distribué (efs) et se connecte à une base de données en cluster. Un CDN permet d'accélérer les requêtes de fichiers statiques.

2.1.3 Avantages

— Démocratisation des bonnes pratiques
— Performances améliorées
— Haute disponibilité

2.2 Erreurs et pannes

2.2.1 Le problème

En cas d'erreur critique sur un serveur ou sur la base de données, de panne inattendue ou d'interruption forcée, il est parfois nécessaire de revenir en arrière, de redéployer le service ou une partie du service en environnement dégradé. Si, en théorie, le P.R.A. (plan de reprise d'activité) doit être testé de façon répétée et travaillé avant la survenue de l'incident, en pratique, il n'est souvent mis à l'épreuve qu'au moment de la survenu d'un problème.

2.2.2 Proposition webcapsule

La webcapsule organise la politique des backups de la base de données. La reprise d'activité se fait en rétablissant la base de données depuis l'un de ces backups et en rétablissant le serveur correspondant, dans un délai minimal. Le PRA est régulièrement testé.
Par ailleurs, pour éviter les pannes critiques et délimiter des redondances suffisantes, une armée des douzes singes sera mise en place.

2.2.3 Avantages

— pas de pertes de données
— nombre de pannes diminuées
— PRA rapide et efficient

2.3 Les secrets

2.3.1 Problème

Un certain nombre de paramètres peuvent ne pas être dans le code git, comme les secrets ou les variables d'environnement, ou certaines dépendances. On souhaite à la fois les garder discrets tout en ne les perdant pas.

2.3.2 Proposition webcapsule

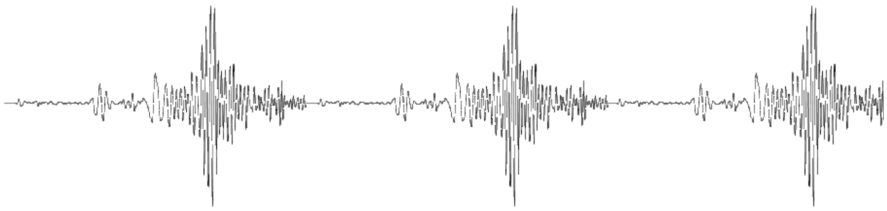
Les environnements, les secrets, les dépendances peuvent être ajoutés et ont un suivi via l'interface graphique. Ils sont gérés le plus discrètement possible

2.3.3 Avantages

— Interface centralisée
— Restriction d'accès efficace
— Données protégées

Étape 3

Lire et voir ; monitoring et alerting



3.1 Indicateurs de suivi

3.1.1 Le problème

Le *debugging* d'une erreur en production est souvent long par manque de log ou de trace disponible. De la même façon, si les performances de l'application se dégradent, il est parfois difficile de cibler précisément la raison de cette dégradation. Un monitoring efficace vise donc à rendre disponible, centralisée et claire les informations d'état de l'application.
Différentes solutions de monitoring existent mais chacune est coûteuse en temps de mise en place.

3.1.2 Proposition webcapsule

La webcapsule est partenaire de *Datadog*, solution la plus aboutie pour le monitoring complet d'une infrastructure. On peut, en quelques graphiques, suivre l'état complet et précis de l'application.
L'état complet de l'application comprend deux aspects : le suivi de l'infrastructure et des serveurs, le suivi des logs et des métriques applicatifs

3.1.3 Avantages

— Remontée d'erreur rapide
— Localisation des bugs efficaces, grâce au tracing d'application
— *Alerting* en temps réel

3.2 Indicateurs de qualité

3.2.1 Le problème

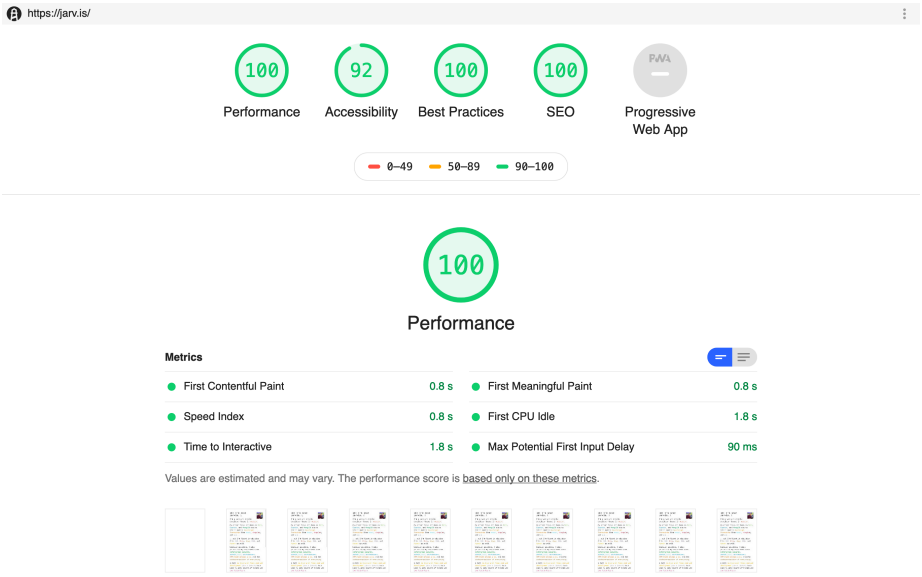
Une fois un site en production, l'on veut parfois connaître son degré d'achèvement, la qualité de sa réponse, sa réactivité ou encore sa compatibilité avec le SEO.

3.2.2 Proposition webcapsule

Webcapsule intègre l'outil *Lighthouse* et permet de générer des rapports. Cet outil indique la qualité du site pour le SEO, ses performances de chargement et propose des points d'amélioration.

Exemple

Un exemple de rapport généré par lighthouse :



L'on dispose des performances du site, d'une métrique pour le SEO et de conseils pour améliorer les différents aspects.

3.2.3 Avantages

— Historique des performances du site
— Données centralisées en un seul lieu

3.3 Mises à jour

3.3.1 Problème

Lors de la vie d'une application, certaines dépendances peuvent devenir obsolètes ou ouvrir des failles de sécurité. Les logiciels principaux eux-même, comme le SGBD ou le runtime du serveur peuvent nécessiter une mise à jour. Il n'est alors souvent pas évident de savoir comment s'y prendre et des retards de version s'accumulent facilement par crainte d'une erreur.

3.3.2 Proposition webcapsule

La webcapsule se présente ici comme une mutuelle des problèmes. Les mises à jour mineures sont effectuées automatiquement. Si des mises à jour majeures apparaissent nécessaires, pour des raisons de sécurité ou de performance, un

email est envoyé, contenant des informations détaillées afin de pouvoir l'effectuer manuellement.

3.3.3 Avantages

— Suivi des dépendances et des versions mutualisé
— Information rapide
— Instruction détaillée